MLS with MILS?

Rance J. DeLong

LynuxWorks

Center for Advanced Study and Practice of Information Assurance

Santa Clara University

Theme

MILS, with an associated ***ecosystem*** of users, vendors and products, ***can provide*** a vehicle to achieve many longstanding information assurance goals, as well as some of the novel security challenges going forward in building the Global Information Grid.

MLS is (at least one of) the goal(s) of MILS.

We must develop a clear, shared vision for how to apply MILS, how to develop systems using MILS, how to tackle the certification challenges, and how to cultivate and shape a vital and fruitful MILS ecosystem.

Success will depend largely on ***how*** we proceed.

Topics

- The MILS premise
- The MLS Goal
- Why do MLS with MILS?
- Systems easily achieved with MILS
- What is MLS and how is it different?
- Subjects, Objects, and Attributes
- Examples of MLS with MILS
- What else is there remaining to do?
- What are the expectations?
- What guidance has been given?
- What MLS solutions are being pursued?
- What is not being addressed?
- What do we really need?

The MILS Premise

- As observed by Rushby in ’81, many security applications are relatively simple single-purpose systems, e.g., downgraders and guards; a ***separation kernel*** can provide a platform for these without the need for a general purpose MLS operating system.
- Extend avionics-style partitioning kernels to include information flow control and added assurance, using partitions as independent security ***domains***. (“MIDS”?)
- Use MILS separation kernels to build, on a single processor, systems normally built as simple distributed systems.
- Provide reference monitor properties of mechanisms to enforce higher-level policies via SK guarantees

MILS and the MLS Goal

- MILS was conceived for security- and safety-critical ***embedded*** systems
- But, many envision it as the foundation for ***general-purpose, high-assurance information systems***, e.g., government and commercial systems at the greatest risk.
- Many such applications entail functional requirements characterized by *multilevel security* (MLS), beyond straightforward MILS

Why do MLS with MILS?

- Traditional, commercially available MLS systems lacked the assurance to be deployed in environments requiring high robustness (where the combination of asset value and threats necessitate the highest levels of assurance)

- MILS promises to provide the needed, and previously largely unattained, level of assurance

So, are we done?

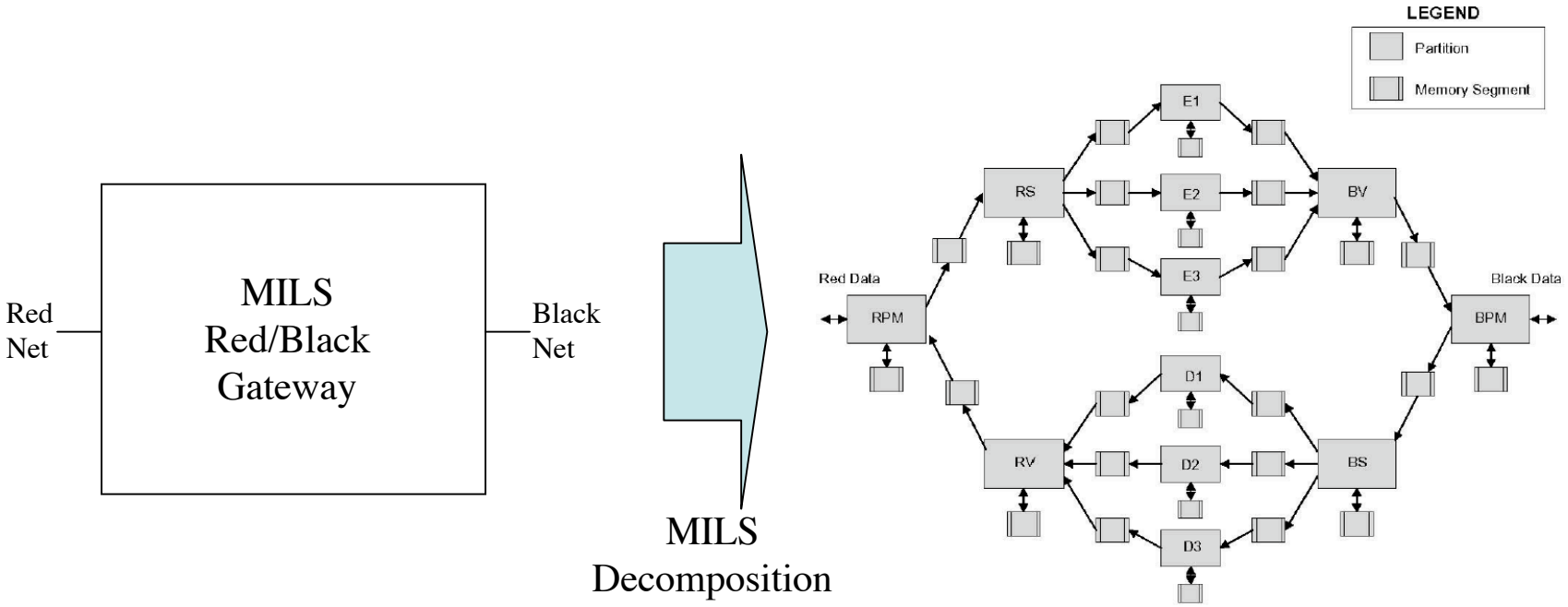
- If MILS is the answer, is there still a problem?

Yes.

- ***How*** will we ***use*** MILS to achieve MLS in all the variations and scale needed in the future?

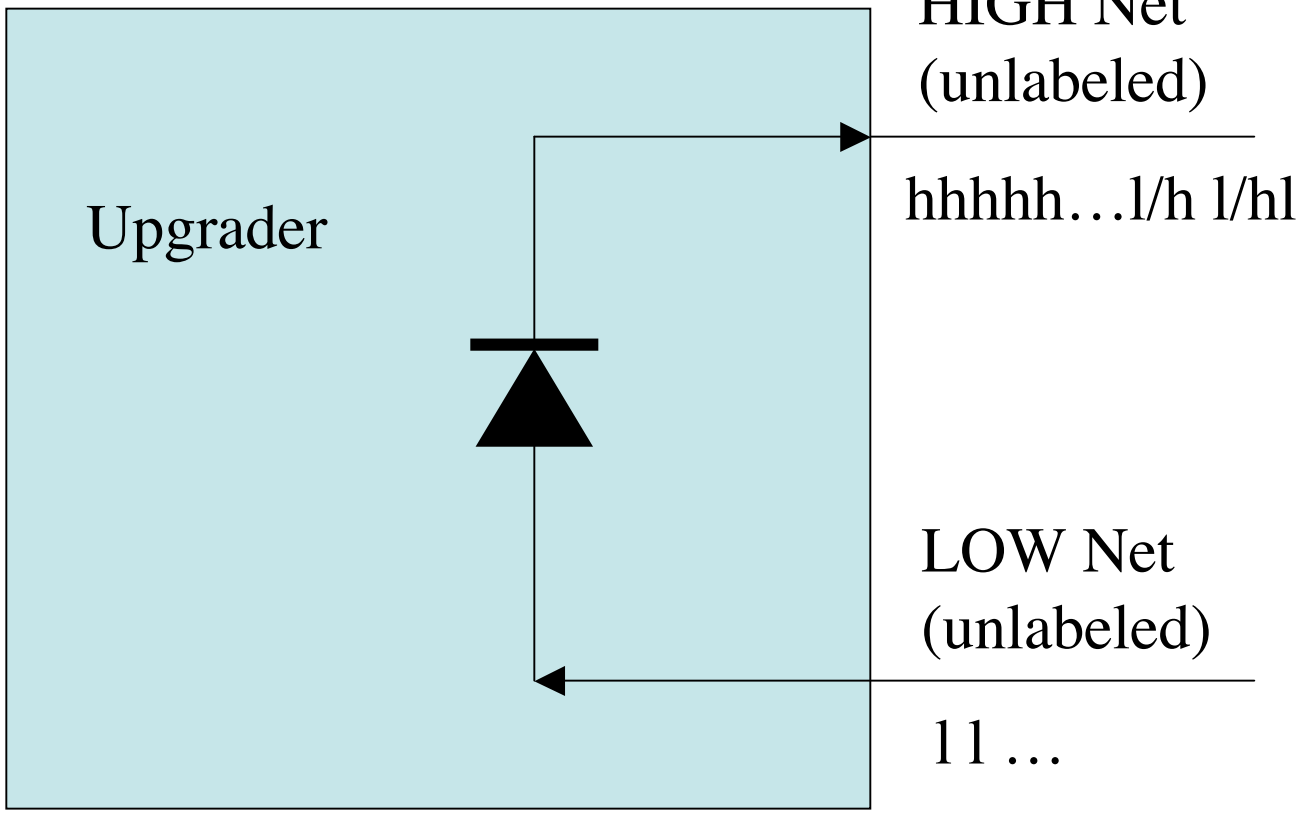
Consider some simple systems easily realizable with MILS

In the following examples we do ***not*** focus on the MILS ***decomposition***, as has been described elsewhere, e.g., the familiar MILS network gateway example:



MSL Example: Upgrader between Multiple Single-Level Networks

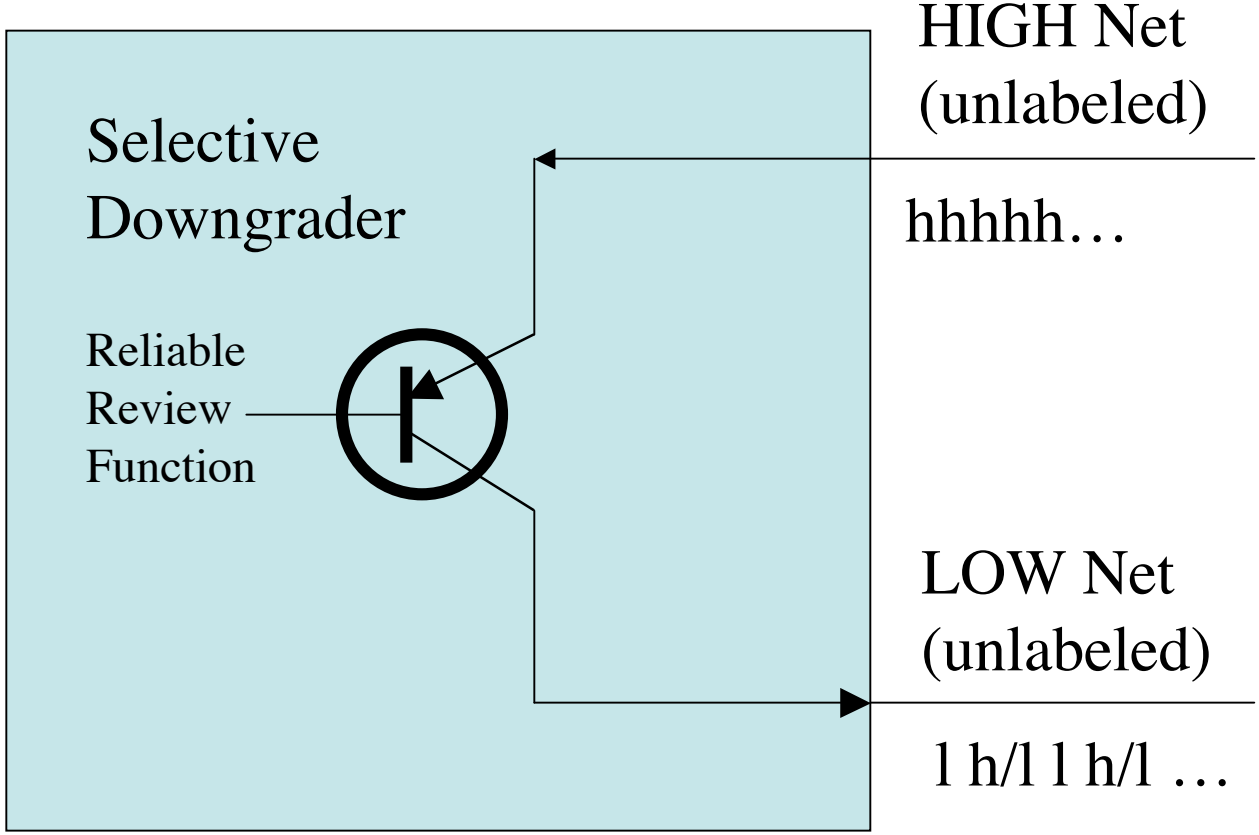
E.g.: NRL Pump provides up flow and manages potential covert timing "back" channel



l/h - low info available at high

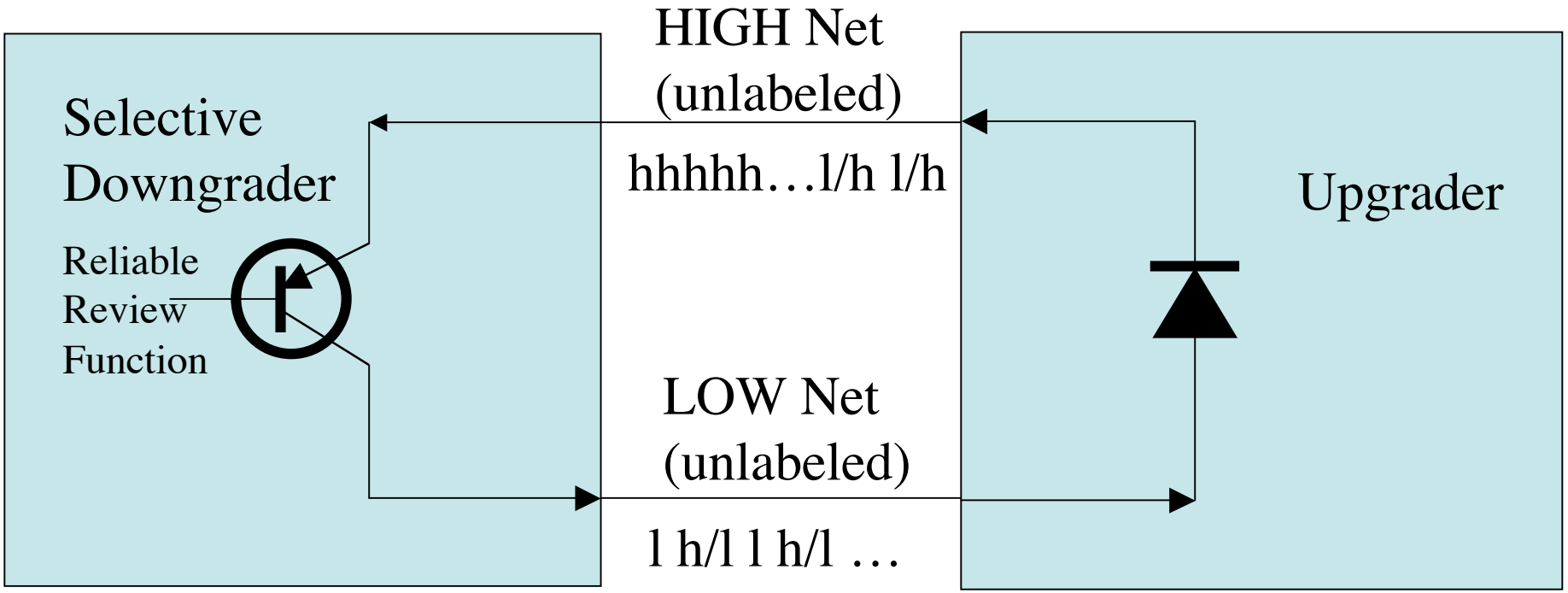
Example: Release Agent between Multiple Single-Level Networks

E.g.: ISSE Guard Checks format and/or content before permitting down flow



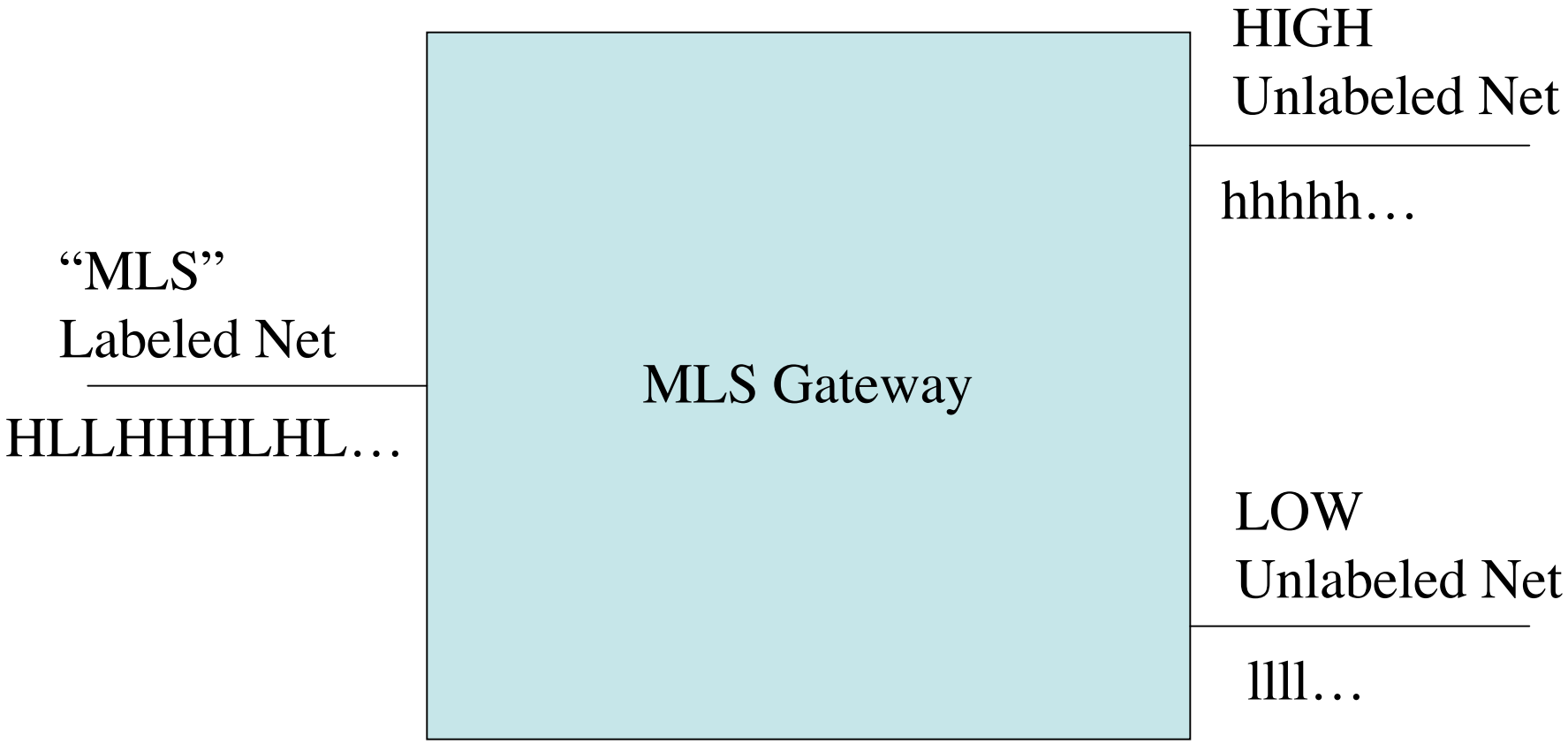
h/l - high info downgraded to low

Example: Combined functions in operational MSL network



l/h - low info available at high
h/l - high info downgraded to low

Example: Labeled MLS network to Unlabeled Single-Level Nets



H, L - explicit labels
h, l - implicit labels

These systems are easily realized with MILS Separation Kernels

- A small number of partitions
- Simple, static information flow policy
- Architecture, instead of labels and MLS policy mechanisms of MLS operating systems
- The information flow policy of the system is ***directly*** implemented by the SK's configured information flows. The SK ***is*** the reference validation mechanism. (Trusted downgrader implements approved downgrading.)

If we know how to build such systems using separation kernels aren't we all but done?

No.

Not until …

- We have a blueprint for building scaleable general-purpose MLS systems that have performance and usability competitive with the untrustworthy systems
- We have familiar APIs, user interfaces, and other services
- We have created the MILS Ecosystem and made it self sustaining

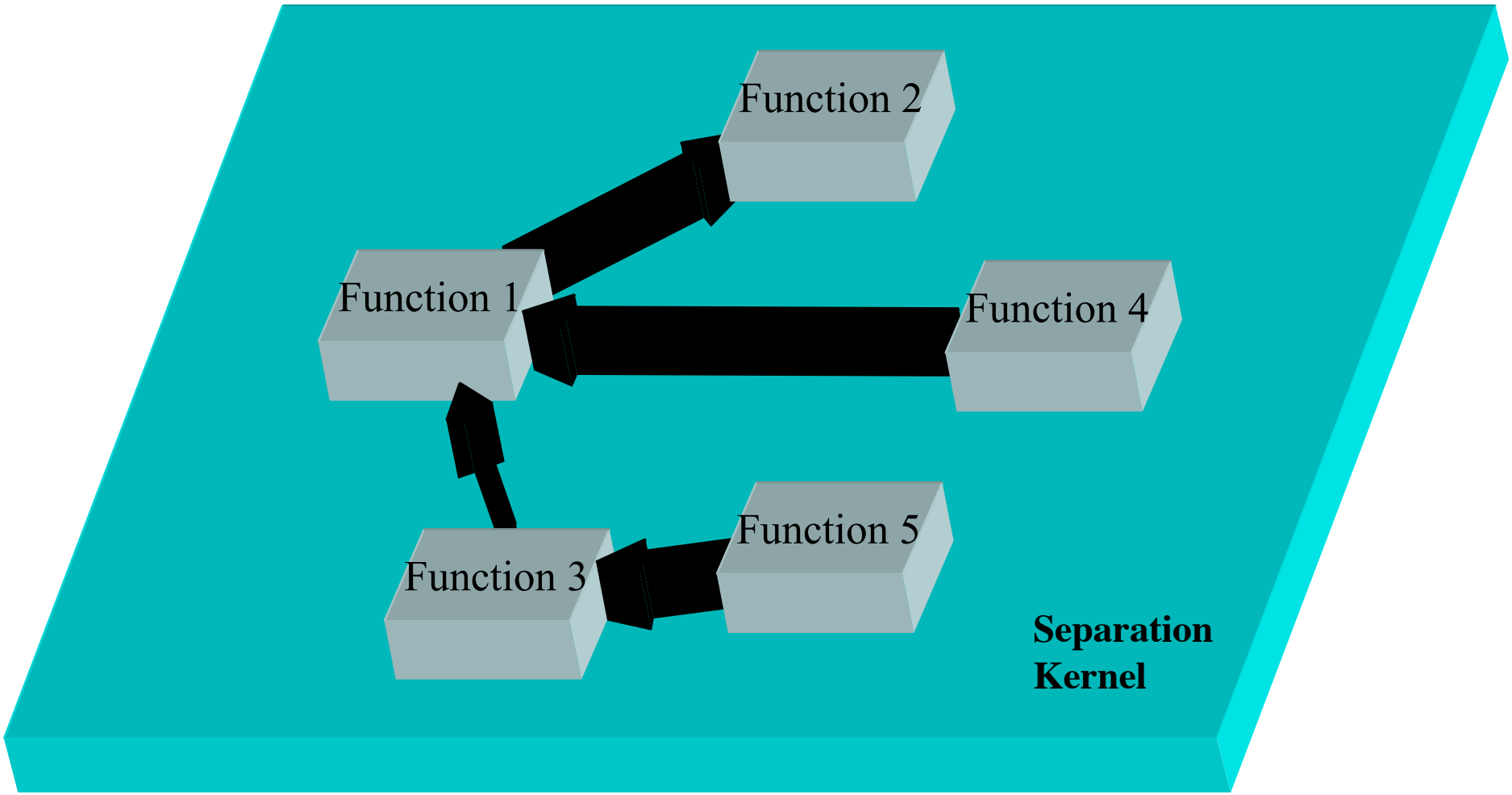
Let's take a step back …

What is MILS?

What is MLS?

What is MILS?

Separation Kernel creates partitions and connections of a distributed architecture



What is MILS?
Separation Kernel *is the "silicon" for secure software architecture*

- It is the ***substrate*** that provides the foundational policies of isolation and information flow control to the architecture
- It may also have other useful foundational properties and functions
- High-assurance subsystems can be built on this substrate to provide secure services
- The possibilities are as endless as those of silicon

What *is* MLS?

- Extant definitions have covered a spectrum:
  - A system having information and users of diverse security levels in which there are authorized users that are not authorized to see some of the information and the system is trusted to not disclose information without authorization
  - A system processing multiple levels of labeled information that is trusted to not permit high information to mix with low information in a way that will result in unauthorized disclosure
  - A system in which high inputs do not influence low outputs

What is MLS?

- The preceding definitions express *different criteria*. You may encounter individuals who fervently embrace one of these positions:
  - "It isn't MLS if it doesn't have ***users***."
  - "It isn't MLS if it doesn't have ***labels***."
  - "It isn't MLS if it doesn't have attribute based ***access controls***."
  - "It isn't MLS if it doesn't restrict ***information flow***."
- What seems straightforward at first becomes difficult as you attempt to be more precise in capturing MLS policy.
- So, is MLS

  access control?

  or

  information flow control?

A common use of the term "MLS"

- "MLS" is popularly used to refer to a ***particular type of system***
  - A system with "security labels" (and, perhaps, integrity labels) applied to "subjects" and "objects" . . .
  - . . . that enforces a mandatory security (integrity) policy having some non-discretionary aspects, e. g., Bell and Lapadula (Biba)
  - E.g., historically:

    A "B1+CMW" MLS operating system / windowing system / network / data base system
- Moral: MLS is a semantic minefield!

MLS access control: Subjects and Objects

- Possible subjects
  - SKPP “partitions”
  - SKPP “subjects”
  - Guest OSen
  - Guest OS processes
  - Threads
- Possible objects
  - SKPP “partitions”
  - SKPP “exported resources”
  - IPC
  - Memory
    - Pages
    - Segments
  - Subjects (passive)
  - Files and Directories
  - “Paragraphs” in a doc
  - Devices

MAC/DAC Access Controls Based on Subject / Object Security Attributes

- Subject Clearance (basis for MAC decisions)
- Integrity (trustworthiness)
- Associated “user” (basis for DAC decisions)

- Object Classification (basis for MAC decisions)
- Integrity (veracity)
- Associated “owner” (basis for DAC decisions)

MLS access control decisions made on the basis of Subject and Object Attributes and an MLS Policy

Binding of attributes

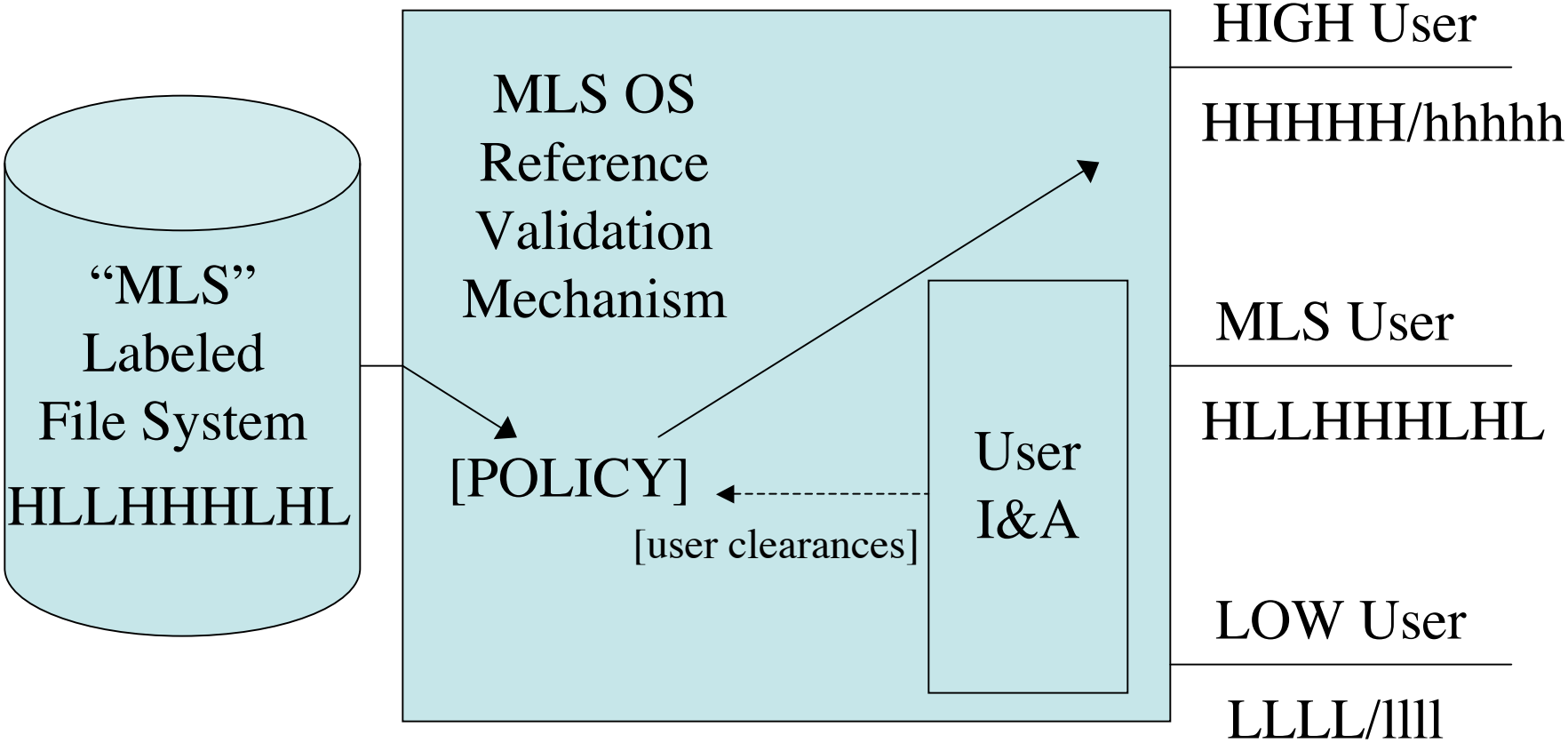
- Association of attributes with subjects/objects
  - Attribute fields vs. attribute values (fields take on values)
  - Binding must be trustworthy!
- Binding time issues
  - Are creation of subjects and/or objects dynamic?
  - Static configuration
    - May permit early binding of unchangable attribute values for a fixed set of subjects and objects
    - Explicit representation of attributes may be unnecessary
  - Dynamic configuration
    - Binding of attribute fields to subjects/objects upon creation
    - Initial values may be permitted to change subject to “tranquility” constraints
    - Explicit representation of attributes may be necessary

Lightweight or Heavyweight MLS?

- Lightweight realization
  - For dedicated service
  - Functionality limited to provided service
  - Small fixed number of security domains (enclaves)
  - Fixed configurations, fixed set of subjects/objects
  - Early binding of [implicit] labels to subjects/objects
  - Architecture embodies at least aspects of the policy

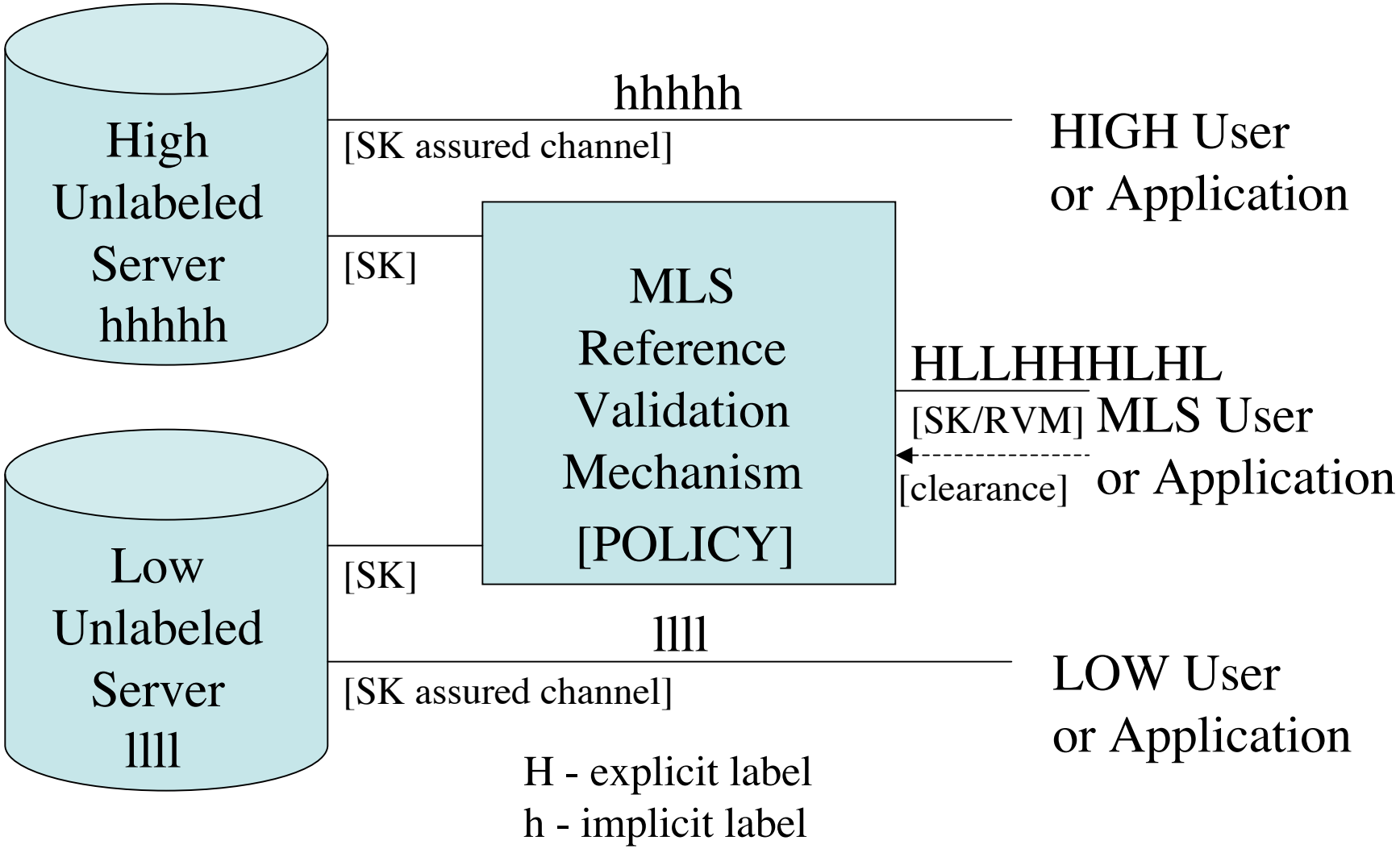
- Heavyweight realization
  - For general purpose computing and information services
  - Full and extensible functionality
  - Potentially large number of security domains, created on demand
  - Dynamic configuration, dynamic creation of subjects/objects
  - Late binding of explicit labels to subjects/objects
  - Reference validation mechanisms enforce policy by labels
  - Architecture supports policy enforcing mechanisms

Example: MLS Operating System
(heavyweight)

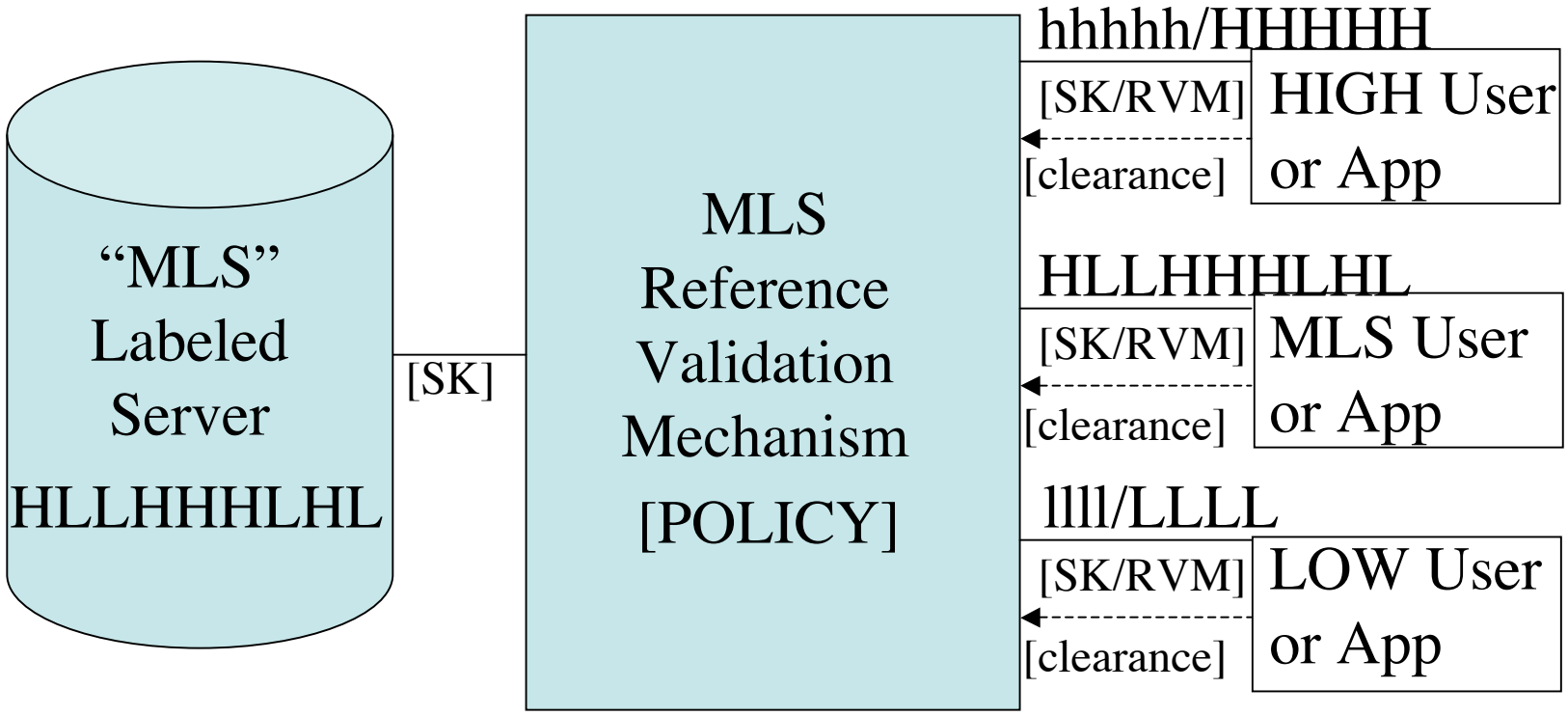


H - explicit label
h - implicit label

Example: Unlabeled Single-Level Servers, MLS RVM (lightweight)



Example: Generalized Object Server - Labeled Server, MLS RVM

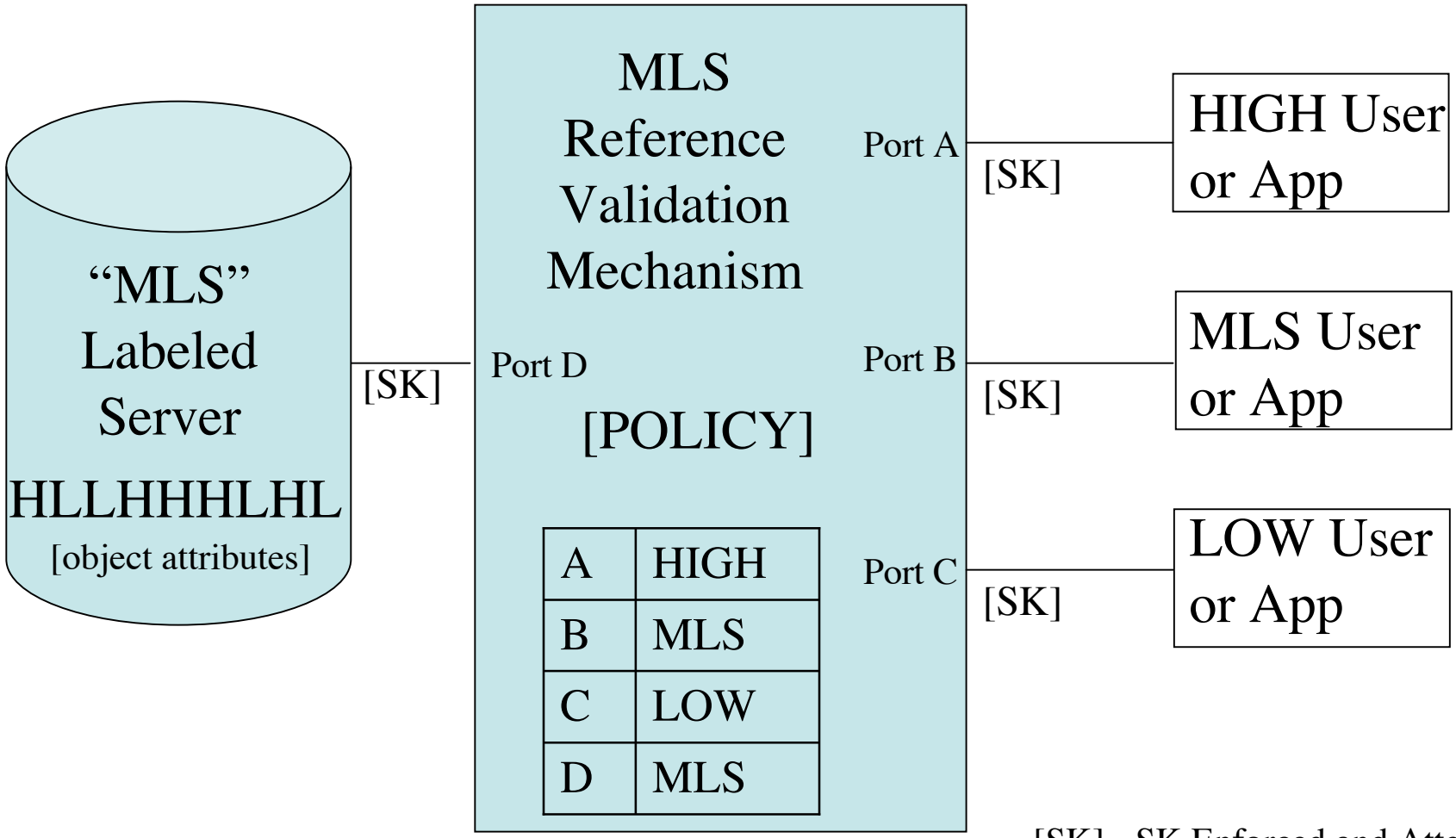


[SK] - SK Enforced and Attested
H - explicit label
h - implicit label

Subj/Obj Attributes: where & how

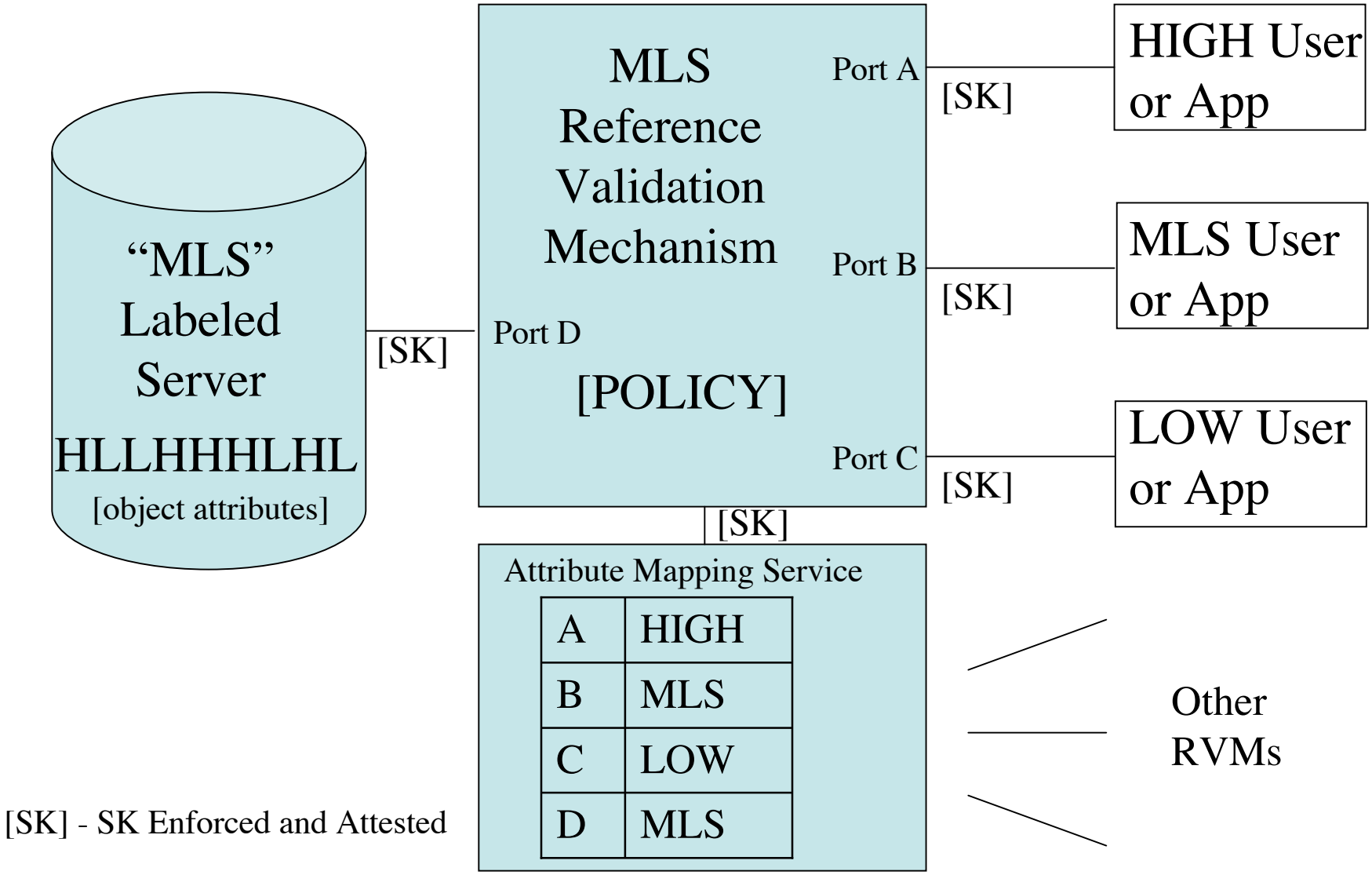
- Association between Subj/Obj and its MLS attributes, e.g., Sensitivity Label, must be reliable, unforgeable, and tamperproof
- For the simplest of static systems can be established directly by the architecture and connectivity
- For special-purpose, static/minimally-dynamic systems with few object types, can be maintained by the RVM
- For general-purpose, dynamic systems with multiple object types, is better maintained by a service separate from the RVM and is available system-wide.
- In any case, the SK supports the attribute bindings and the RVM with non-bypassable communications and tamperproofness

Example:
RVM-maintained subject attributes



[SK] - SK Enforced and Attested

Example: Attribute mapping service



What else?
There is much to do …

- Examples highlight the need for an ***architecture***
- Products should have limited, well defined functions within the architecture
- Architecture should provide certain guarantees and requirements for the component products
- We need to work toward one or more standardized ***architecture templates*** so that product ***Protection Profiles*** and ***Interoperability Profiles*** may be developed

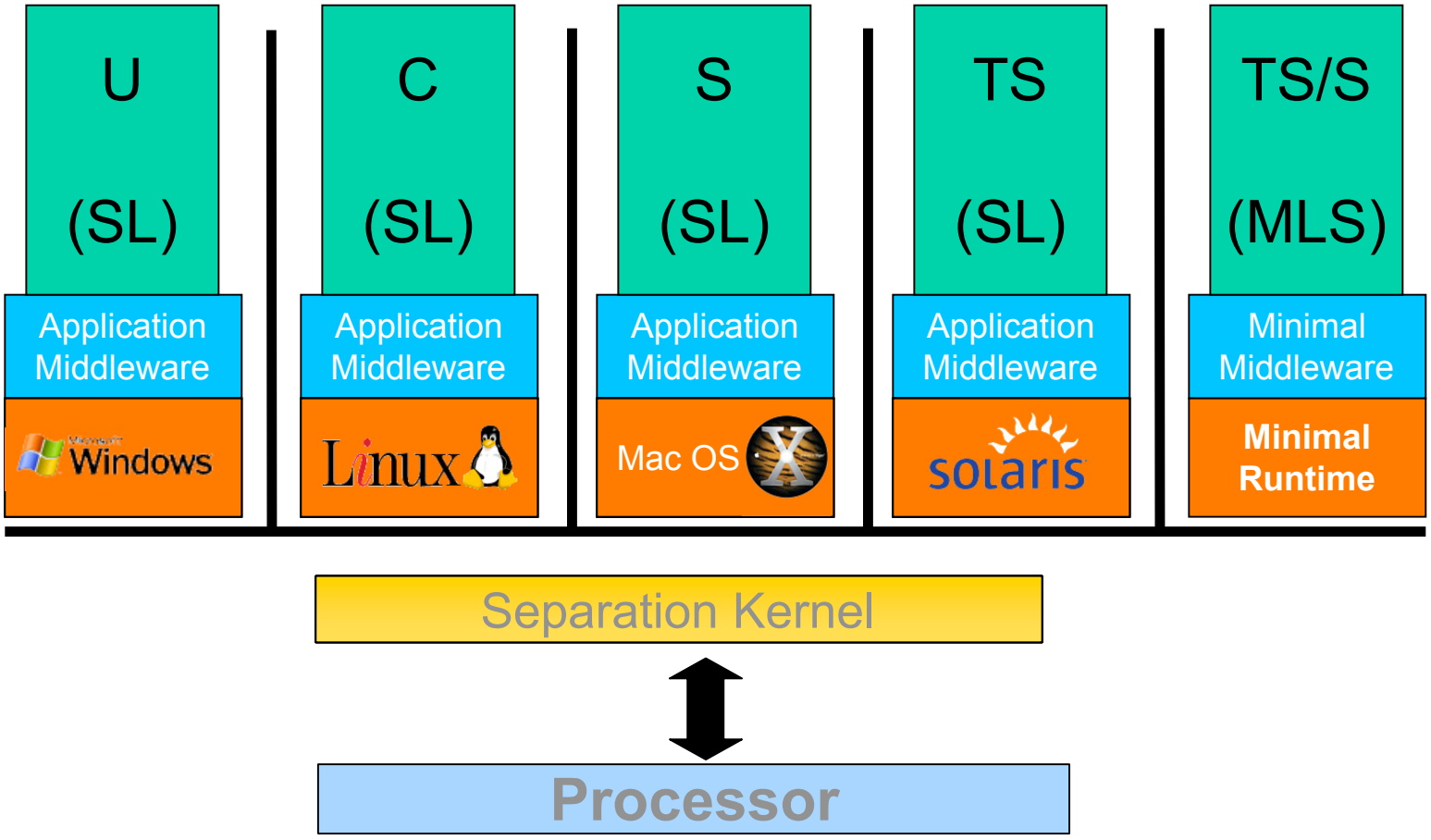
What else?
There is much to do …

- Standard API's for development of trusted subsystems
- Promote methods and tools to facilitate the difficult task of developing high-assurance trusted subsystems
  - ***Formal methodologies and tools*** that can be applied by the ***developers*** while building high-assurance applications
  - ***Development frameworks*** that capture the evidence needed for certification
  - ***Integration frameworks*** to provide the formal underpinnings for chosen architectures
- Cultivate the ***MILS Ecosystem***

Typical proposed MLS “architecture” on MILS:

Guest Operating Systems

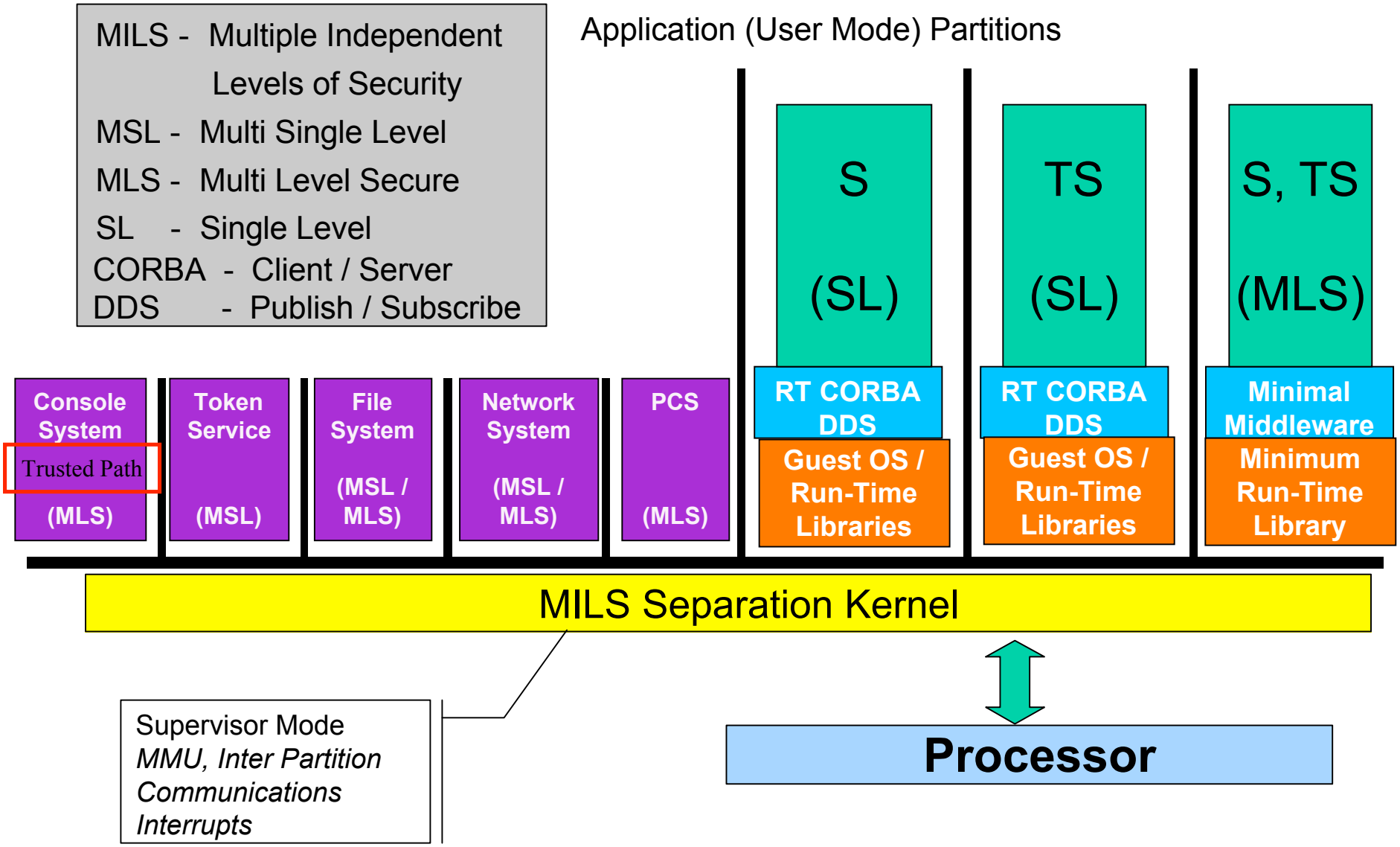
Can you say “MSL”?



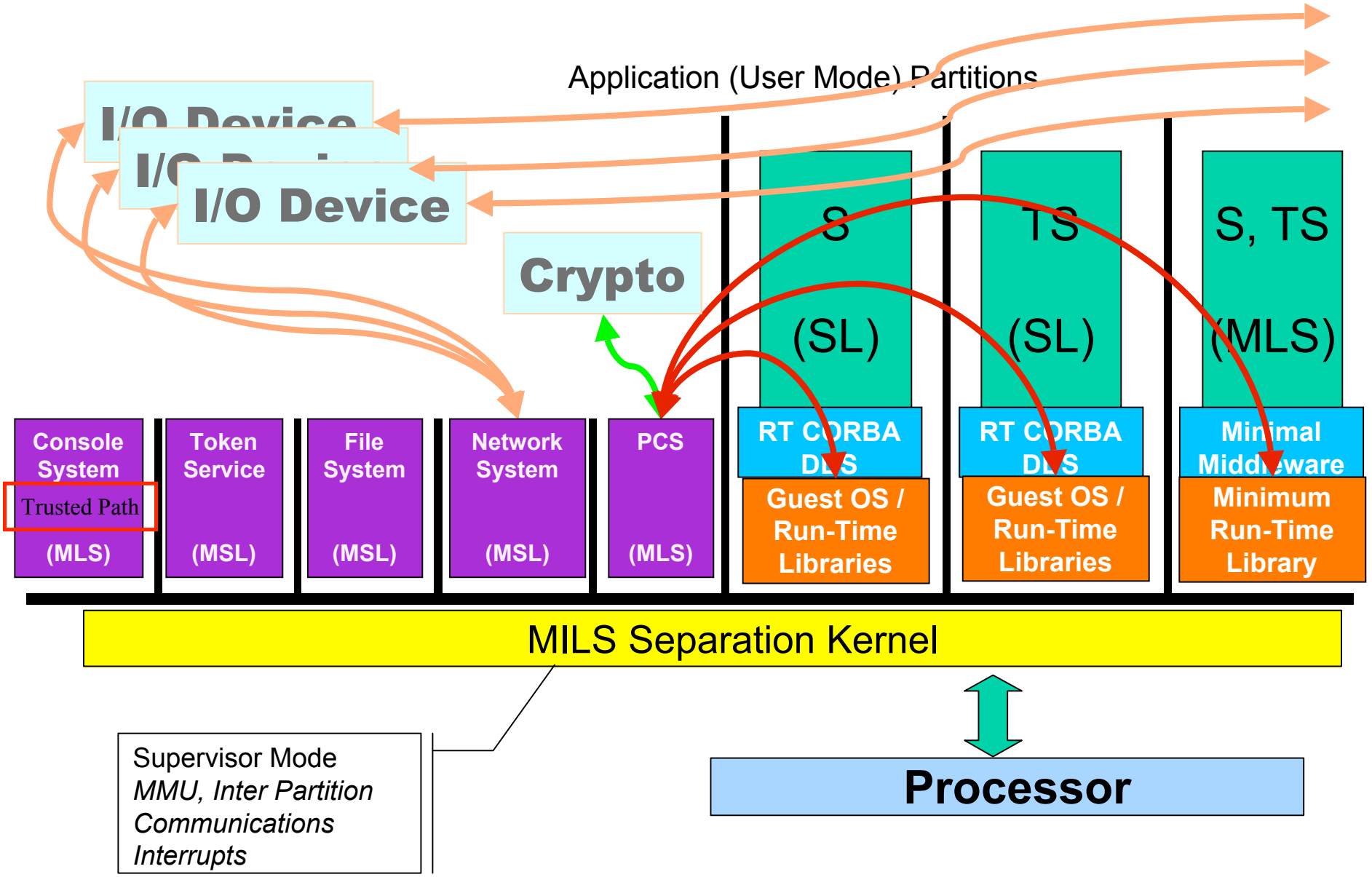
What MLS solutions are being pursued?

- Low- to medium-assurance OSes on VMMs
  - Windows, Linux in VMs
  - Dubious-assurance communications among VMs
  - MSL warmed over
- Ad hoc "MLS" on separation kernels
  - Strong partition separation and information flow control
  - Simple high assurance applications
  - Implicit labels, explicit information flow
  - Ideal for simple, static embedded systems
- These do not cover the needed cases

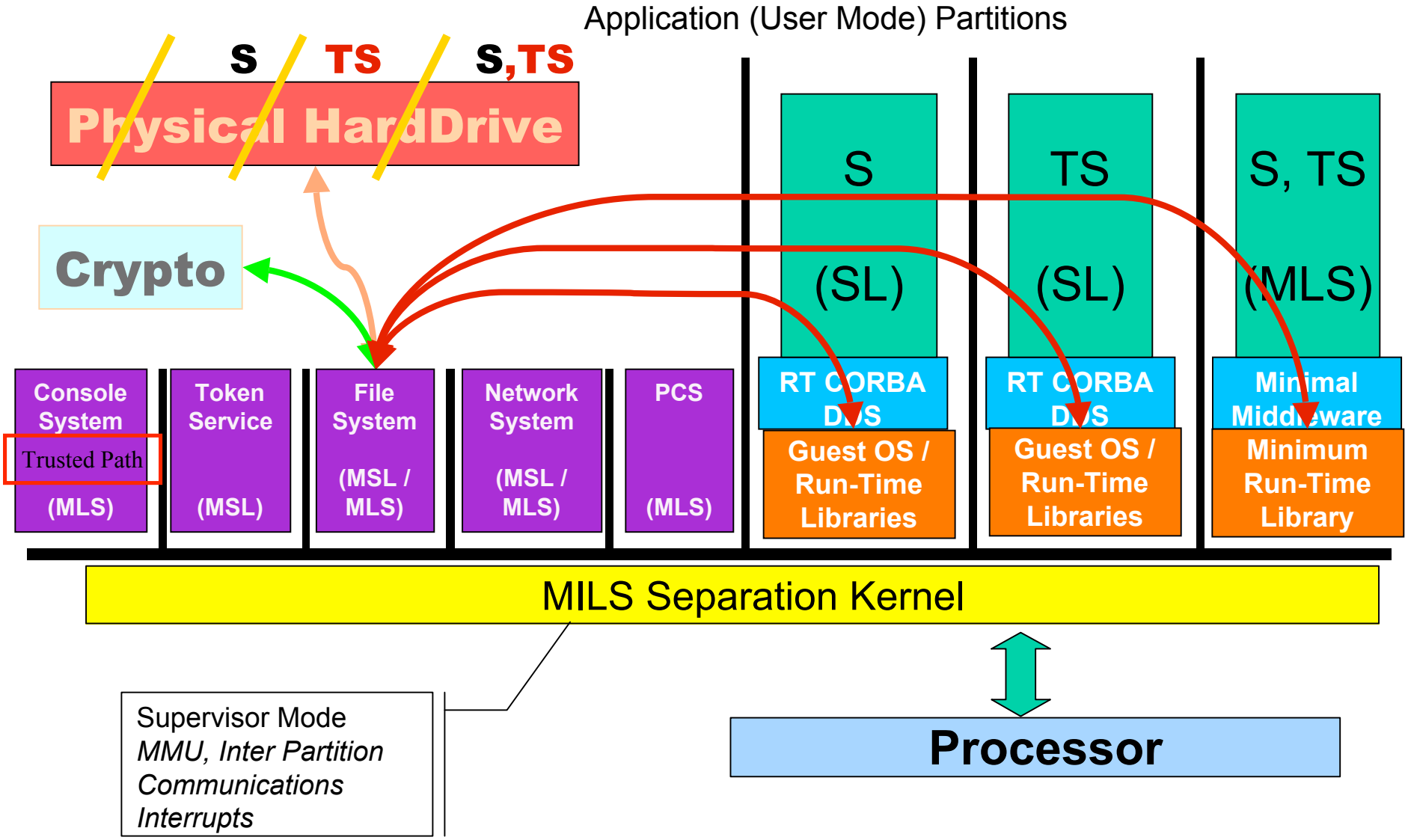
High Assurance MLS Workstation



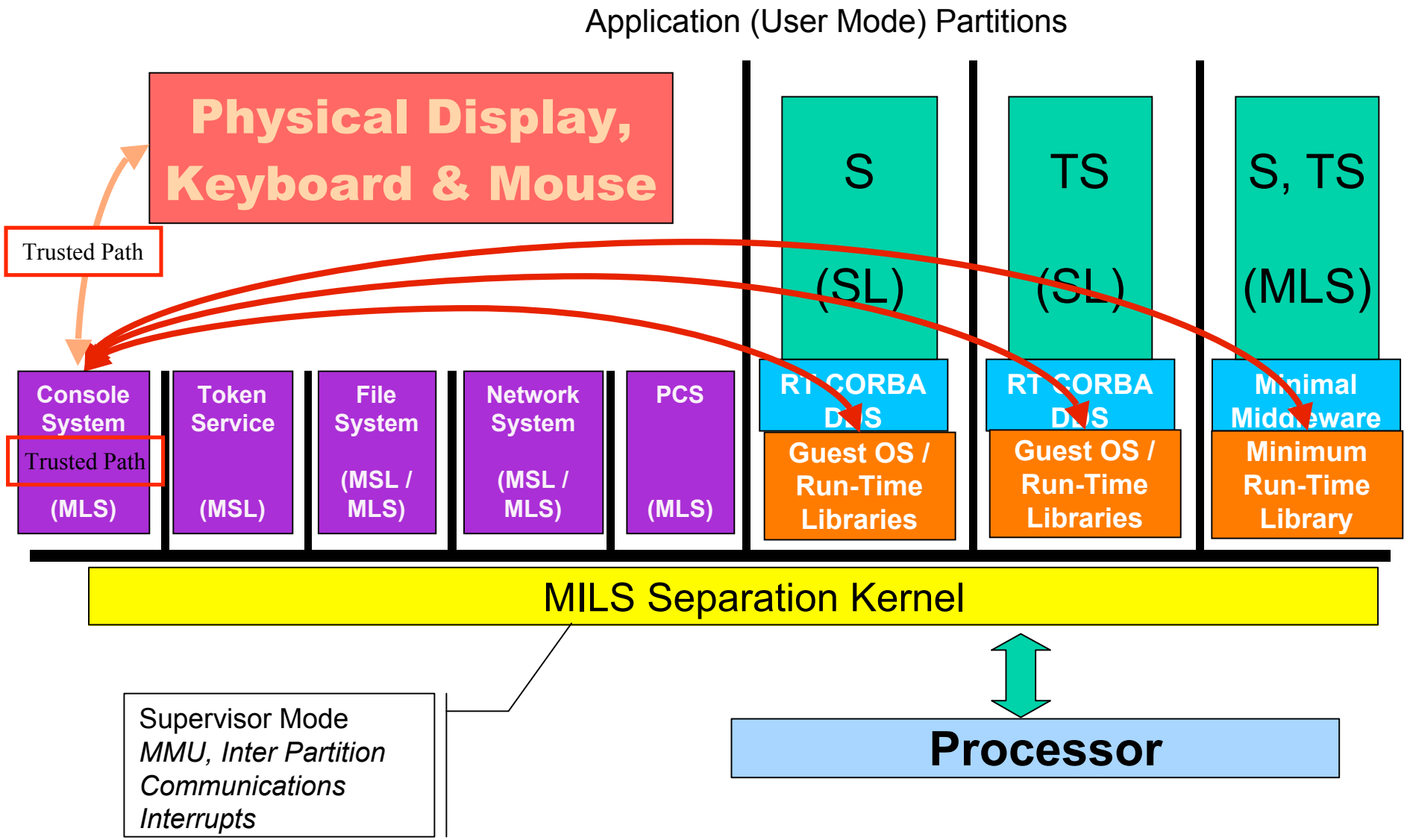
MLS Workstation: Network Access



MILS Workstation: Disk Access



MILS Workstation: HMI Access



What guidance has been given?

- Embedded systems
- Consider most stringent requirement scenarios
  - High-value assets
  - Skilled, motivated attacker with ample resources
  - Multiple levels of separation
    - TS/SCI - U (3 levels)
    - S - U (2 levels)
  - Cleared and uncleared users
  - High physical risk (combat environment)
  - Connectivity (NIPRnet (to Internet), SIPRnet, etc.)
  - System needs to do cross-domain data transfers
- Single component must protect most valuable assets from the most capable attackers under the most difficult circumstances.
- Recommendation: “Due to … risks and the current ability to construct complex systems … refrain from implementing a systems with similar characteristics.”

What is not being addressed?

- High-assurance, full-featured MLS workstations and servers for C^4ISR, and tactical derivatives
- This has been ***promised*** by the promoters of MILS (at least by failing to say otherwise)
- It has been ***understood*** by many to whom MILS has been promoted
- It is ***not*** in the vision of many product developers
- But it ***can be achieved*** provided the community adopts a shared vision and proceeds deliberately to establish the conditions for success

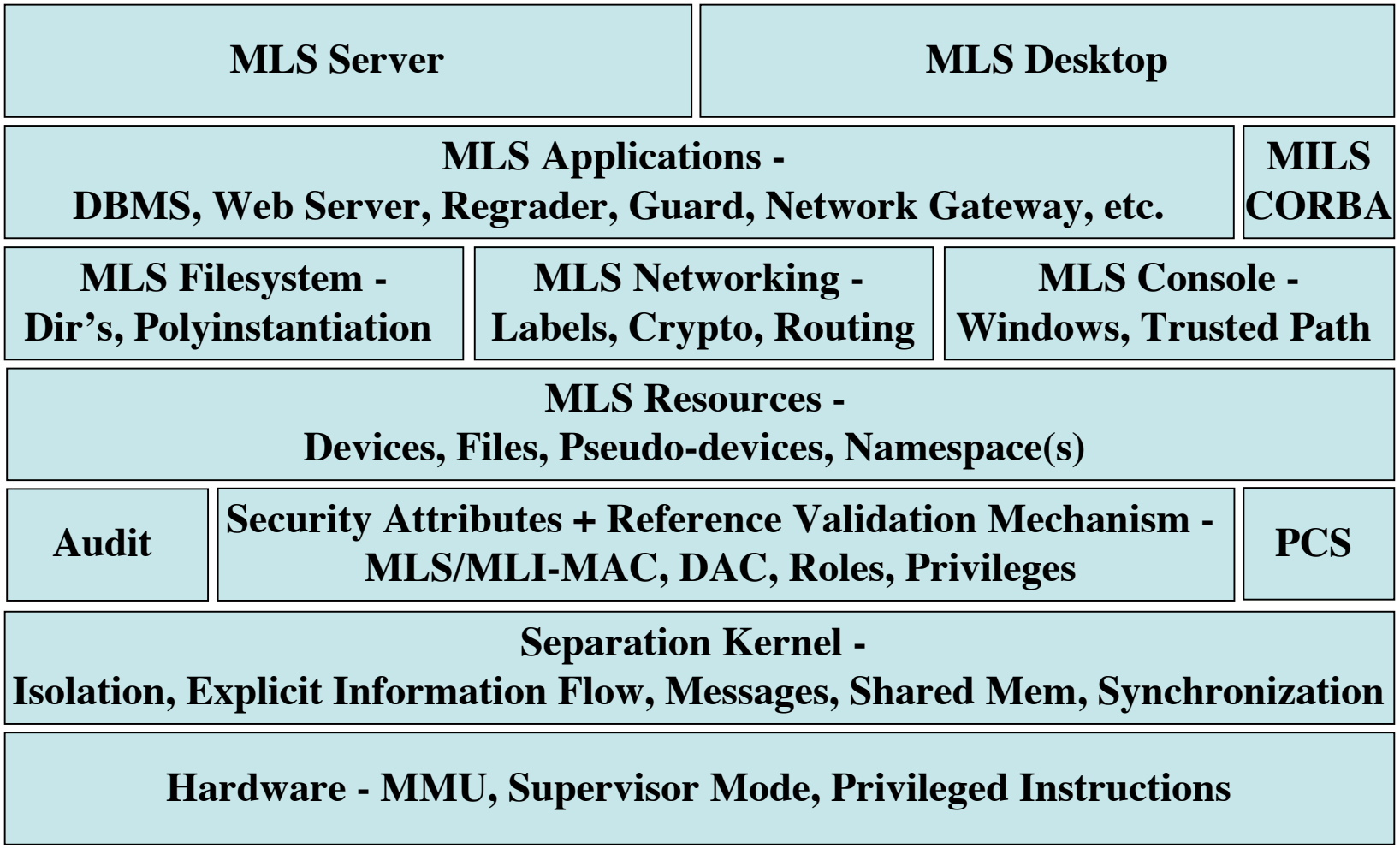
Challenges Inherent in Applying MILS Paradigm

- Scarcity of examples
- Risk of divergence
- Technology selection - what techniques and tools
- Technology integration - how to combine the technologies
- Prevailing bottom-up versus top-down design - it's been bottom-up so far, we recognize the need to do some top-down
- Reusable components - to achieve this we need to address the challenges above, and others
  - Interoperability
  - Compositional Assurance and Evaluation

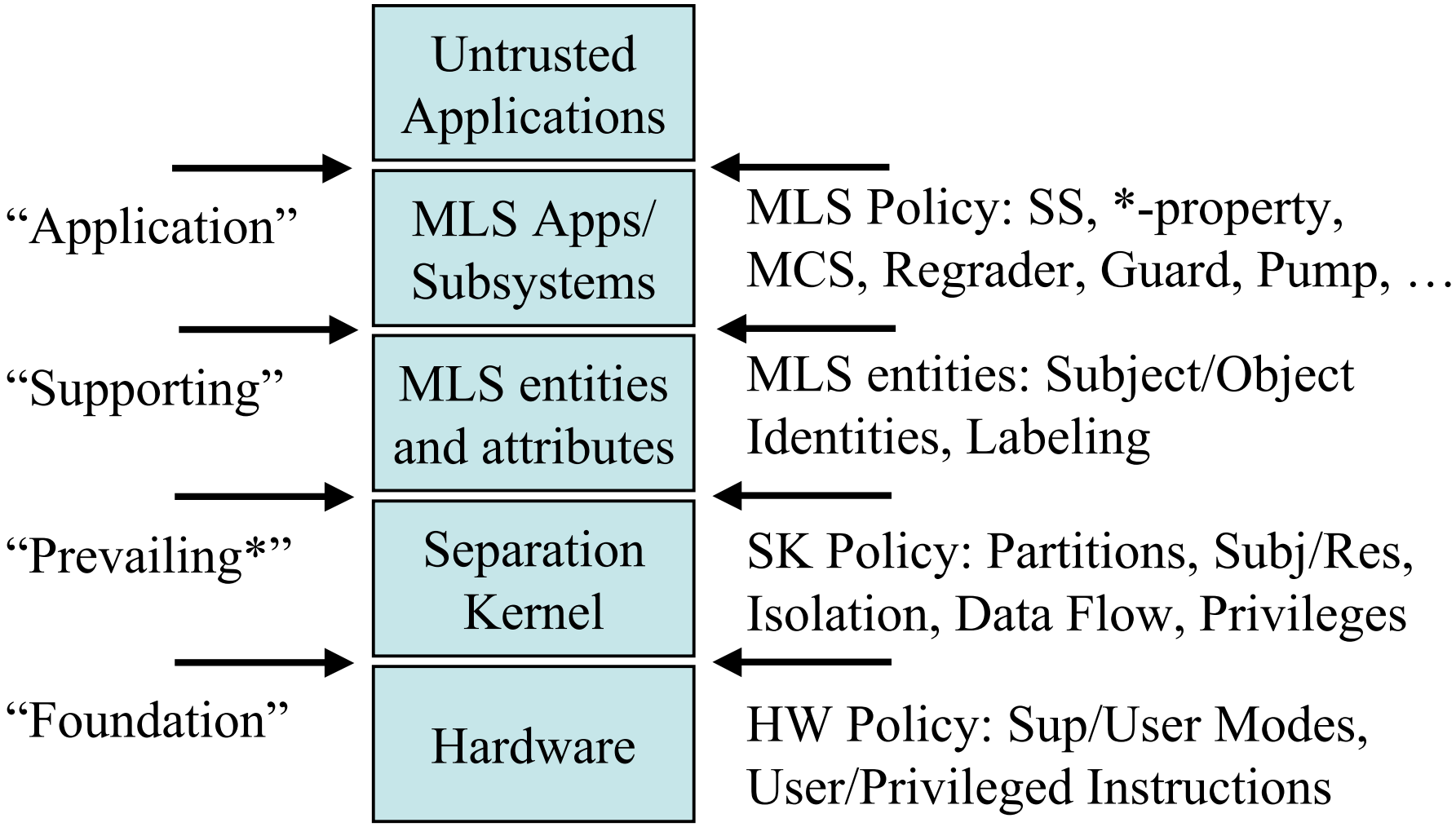
What we need for a high-assurance MILS-based MLS Workstation . . .

- We've got MILS SK, PCS, DDS, CORBA, guest OS, POSIX . . .
- We'll need some other high-assurance MILS subsystems:
  - Console with trusted window system
  - Trusted naming service, identity/integrity attestation
  - Trusted disk storage and filesystems
  - Trusted networking
  - Session management (command env, session lock/unlock, suspend/resume)
  - Application management (dynamic instantiation, dynamic resource mgmt)
  - System management (user admin, app admin, dev mgmt, sys update, plugins)
  - System operations management
  - System self-test, integrity and recovery
  - Auditing (daemon, storage, configuration, analysis)
  - Security management (user/group sec attrs, RBAC, label encoding admin)
  - MLS objects, attributes and policy arbiter (label interpretation)
  - User IAAA - Identification, Authentication, Authorization, Accounting
  - Cryptographic support
  - Generic regrader (rule-driven, type-driven)
  - Daemons (system log, printer, mail)
  - Hardware (elim DMA vulnerabilities, trusted USB controller, graphics devs)

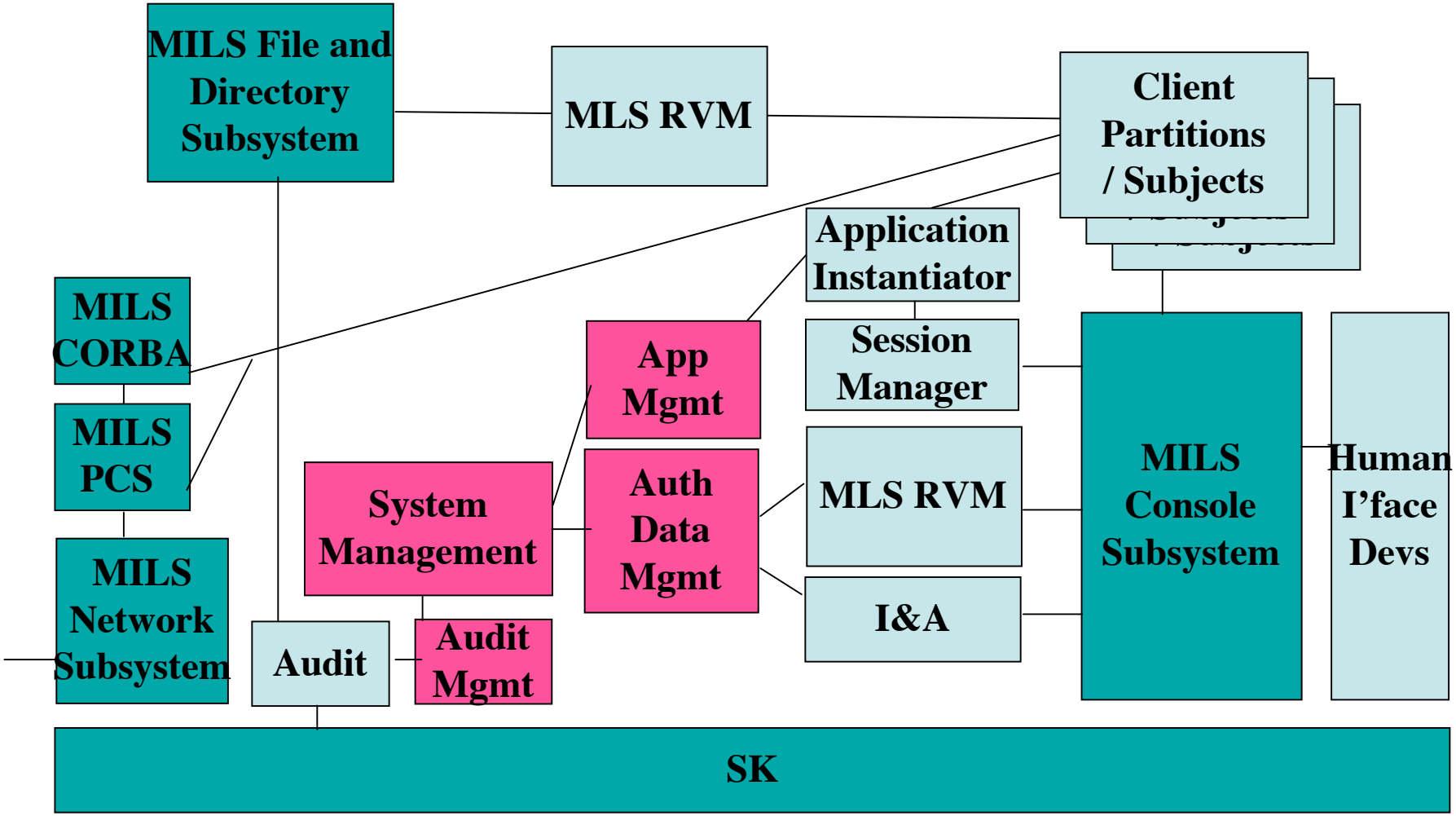
Notional MLS "Software Stack"



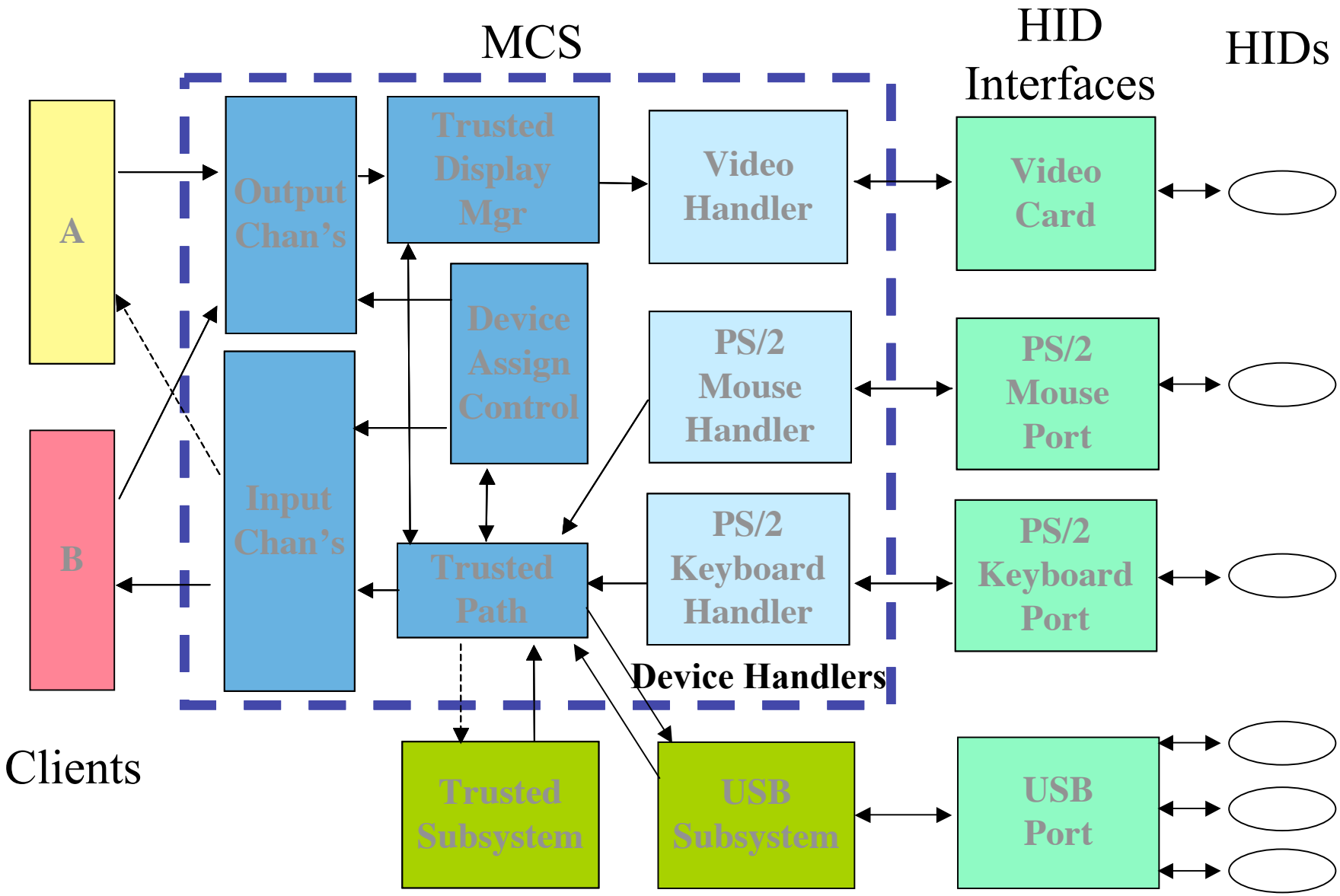
Policy Hierarchy



Illustrative (partial) Architecture of a MILS/MLS workstation



MCS Functional Block Diagram



What do we really need?

- All the pieces
- A unifying architectural approach
- An appropriate set of standards
- Protection Profiles
- Interoperability Profiles
- Development support environments
- Formal integration frameworks